Developing a Standard Interface for Drones

Tully Foote

Open Source Robotics Foundation

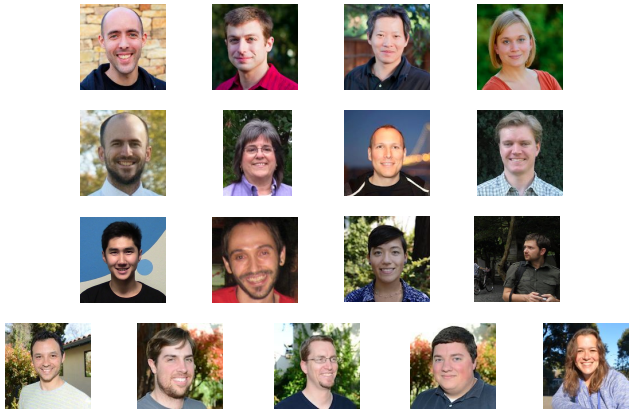
Goals of this talk

- Convince you that this is important
- Provide examples of good interface design
- Give an suggested interface to kickstart the discussion

My Background

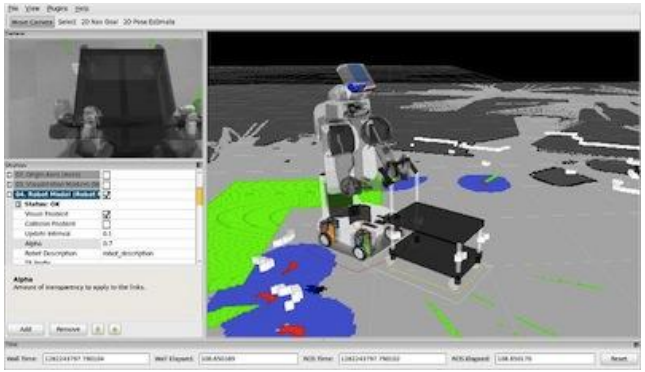
Open Source Robotics Foundation

Mission Statement: “...to support the development, distribution, and adoption of open source software for use in robotics research, education, and product development.”



http://osrfoundation.org

Open Source Robotics Foundation



Open Source Robotics Foundation

My Roles in the ROS Project

ROS Platform Manager

- Core contributor to all 8 major ROS releases

Core developer

- Several packages including many message packages such as sensor_msgs and geometry_msgs

The importance of standard interfaces

The importance of standard interfaces

- They allow interoperability for projects
- They decouple development of modules

asctec_mav_framework	mavlink2ros https://github.com/posilva/mav2rosgenerator
mavros	roscopter https://code.google.com/p/roscopter/
CRATES https://bitbucket.org/asymingt/crates	rospilot
hector_quadrotor (optionally with hector_slam)	autopilot_bridge https://github.com/mikeclement/autopilot_bridge
mav_tools	

Canonical Message Set

Errors fixable via engineering or implementation

Canonical Message Set
What to communicate

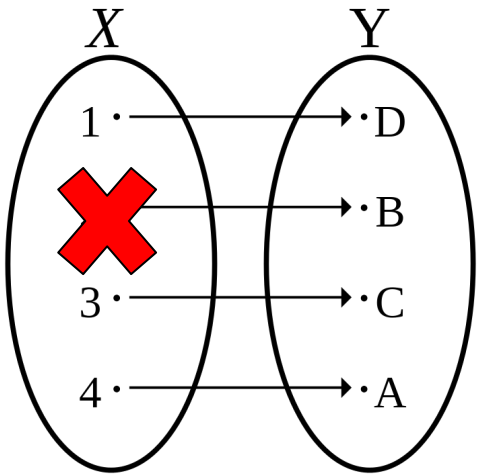
Message Format & Definition
Agreement on how to pack date so someone else can unpack the data reliably.

Transport
How to get the packed data from point A to point B

A standard interface provides:

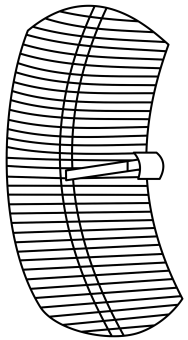
A one-to-one mapping any different representation

If a source is missing it must be approximated, guessed, or manually generated.



Example:

Consider tracking a drone with an active antenna that points in the cardinal direction of the drone.



Can you reuse that signal if you upgrade to a higher gain antenna with heading and azimuth tracking?

Design Guidelines with Examples

Focus on core interfaces

Interfaces should not be burdensome and limiting

Example of too burdensome

```
uint32 MAX_BAT_COUNT=4
uint32 MAX_BAT_REG=48
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
int32 id
int32 lastTimeSystem
uint16 timeLeft
uint16 averageCharge
string message
int32 lastTimeController
uint16 present
uint16 charging
uint16 discharging
uint16 reserved
uint16 powerPresent
uint16 powerNG
uint16 inhibited
pr2_msgs/BatteryState[] battery
  int32 lastTimeBattery
  uint16[48] batReg
  uint16[48] batRegFlag
  int32[48] batRegTime
```

Find the right level of abstraction

Too generic -> not useful, overhead

Too specific -> cannot be reused

It depends on the use case to determine what is the most efficient level of abstraction.

Too Generic: [Int32]MultiArray Message

```
std_msgs/MultiArrayLayout layout
  std_msgs/MultiArrayDimension[] dim
    string label
    uint32 size
    uint32 stride
  uint32 data_offset
int32[] data
```

Leads to complex indexing for users like:
multiarray(i,j,k) = data[data_offset +
dim_stride[1]*i + dim_stride[2]*j + k]

Example Too Specific: PointCloud

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Point32[] points
  float32 x
  float32 y
  float32 z
sensor_msgs/ChannelFloat32[] channels
  string name
  float32[] values
```

Final solution “Just Right”: PointCloud2

```
std_msgs/Header header
  {uint32 seq, time stamp, string frame_id}
uint32 height
uint32 width
sensor_msgs/PointField[] fields
  uint8 INT8=1 uint8 UINT8=2 uint8 INT16=3 uint8 UINT16=4
  uint8 INT32=5 uint8 UINT32=6 uint8 FLOAT32=7 uint8 FLOAT64=8
  {string name, uint32 offset, uint8 datatype, uint32 count }
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Or at least good enough.

Self contained

A self contained message can be:

- Recorded + played back
- Forwarded/remapped
- Delayed in delivery
  - Caching/store and forward
  - Network delays
- Rendered for display

Example Laser Scan

std_msgs/Header header

{uint32 seq, time stamp, string frame_id }

float32 angle_min

float32 angle_max

float32 angle_increment

float32 time_increment

float32 scan_time

float32 range_min

float32 range_max

float32[] ranges

float32[] intensities

High Level Design Feedback

Common complaints

Generality adds overhead:

- Bandwidth
- Complexity

Don’t be penny wise and pound foolish.

Tips for good design

- Focus on the fundamentals of the communication/application
- Keep in mind different use cases for the interface
- Include foreseeable future use cases
- Don’t be stingy on high width data at low frequency.
- It’s important to try things out
- It’s ok to make a mistake, it can be fixed in a new version

Tips for good design

- Units are important!
- Clear documentation is important
- Clearly scope the design
  - It should stand alone
  - There may be uses cases where it can be used more effectively with additional parallel interfaces.
- Don’t try to require everything to be a standard.
  - If something becomes more common then standardize it.

An example of the process for a Drone Interface

Identify the use case

What is universal to all drones? Basic flight control

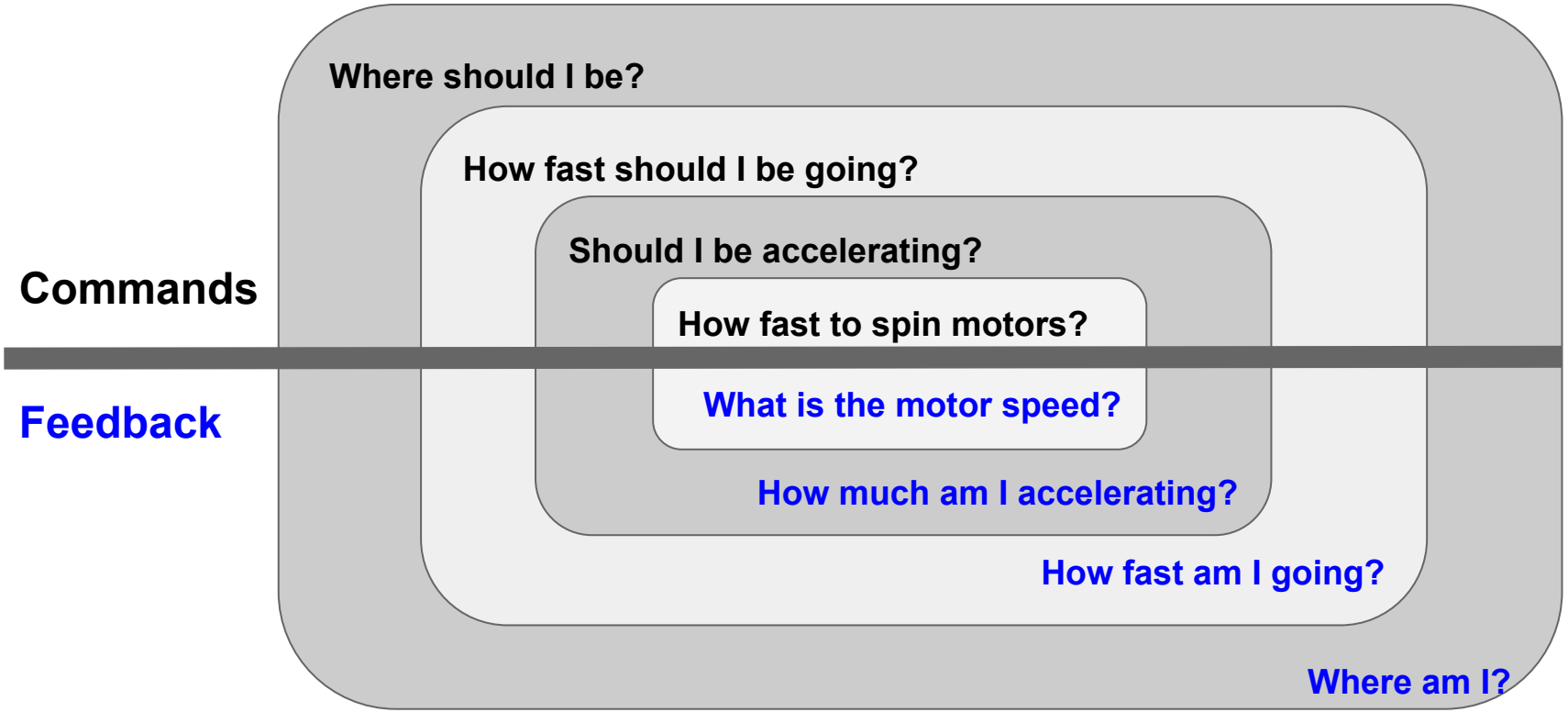
- Flying along a path (maybe zero length)
  - Lower level controls (velocity and acceleration)
- Localization + odometry

There are many higher level abstractions, we'll scope them out for now.

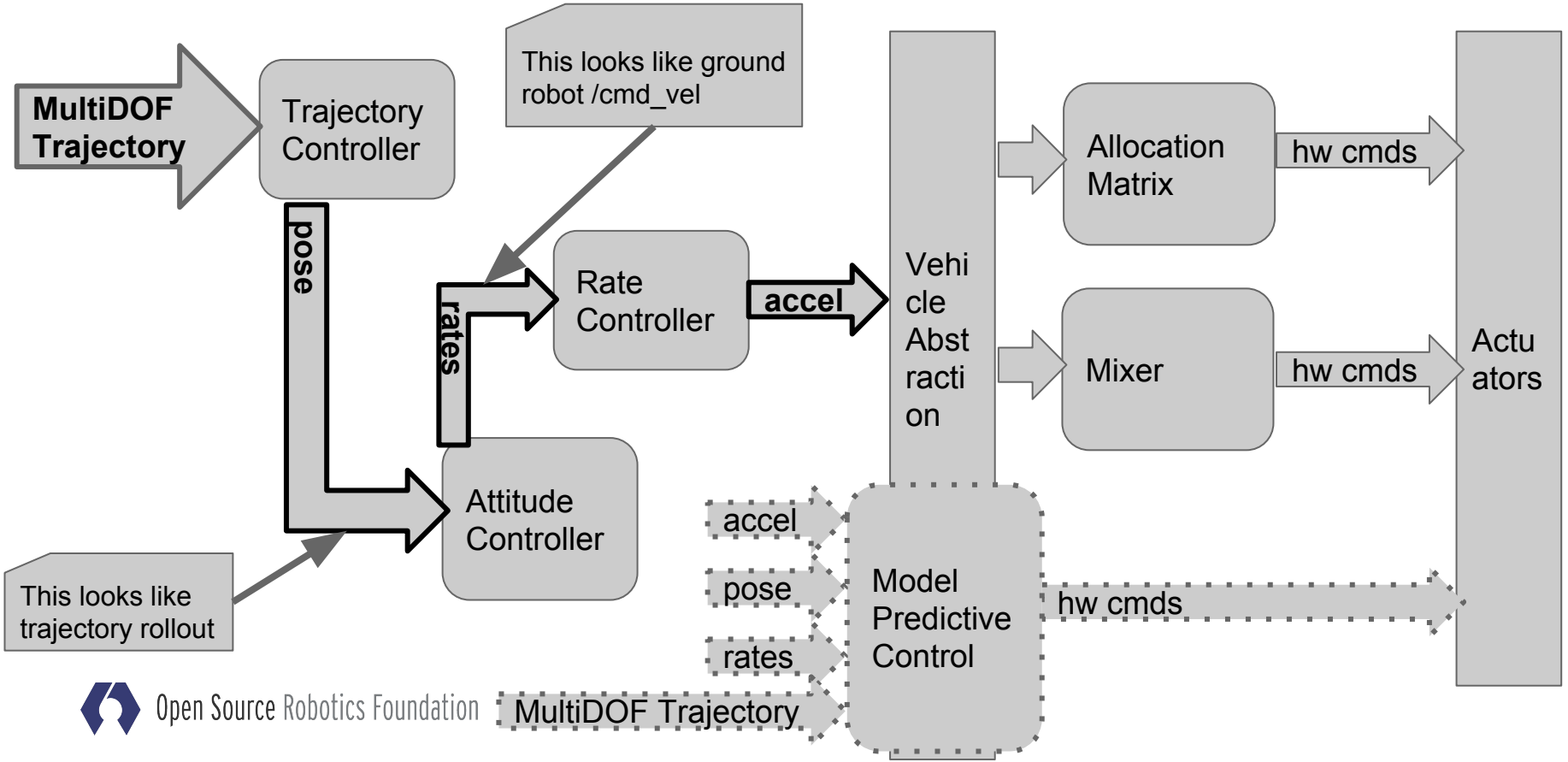
Research existing definitions to adapt or adopt

- Mavlink
- Mavlink2
- ETHZ mav_msgs
- DroneKit
- mavros
- trajectory_msgs
- nav_msgs

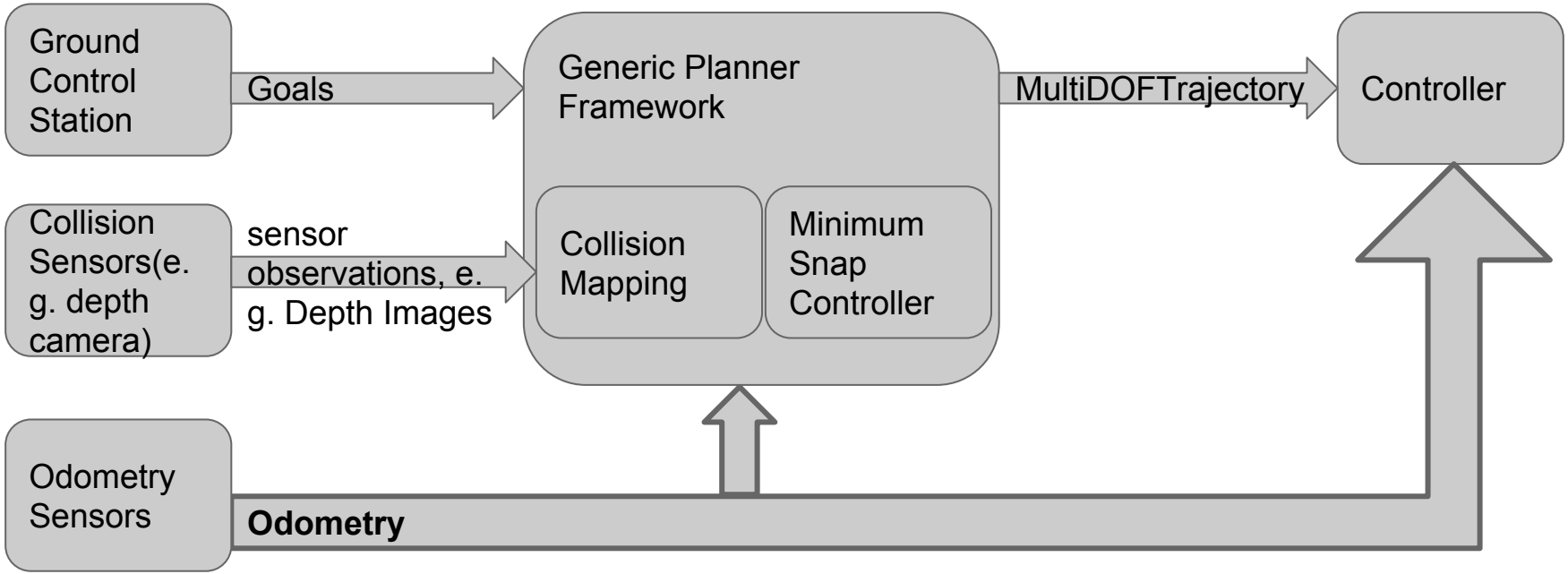
Identify subgroups or connected interfaces



Command Abstractions

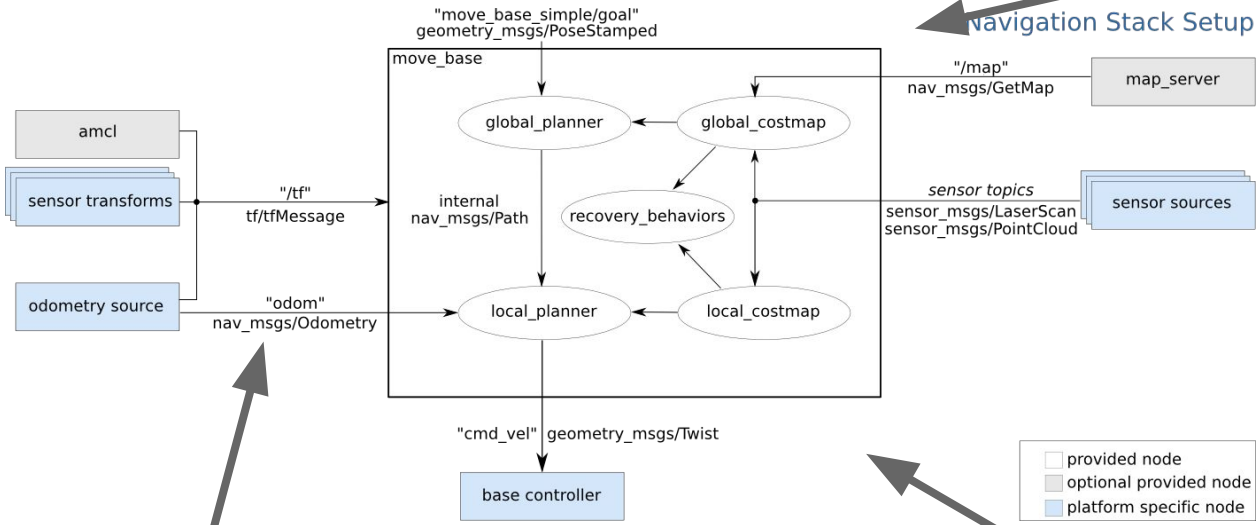


High Level Abstractions



Identify Similar Interfaces

From Ground Robots:



Where should I be?

Where am I?

How fast should I be going?

Proposed Standard Messages for Flying

Proposed Standard Messages for Flying

Represent:

- Path commands with trajectory_msgs/MultiDOFJointTrajectories
- Goal Pose commands with geometry_msgs/PoseStamped
- Odometry with nav_msgs/Odometry extended to add acceleration

Proposed Standard Messages for Flying

Represent:

- Velocity via mav_msgs/AttitudeThrust and mav_msgs/RollPitchYawrateThrust
- Acceleration via mav_msgs/RateThrust

Paths with MultiDOFJointTrajectory

Pros:

- Existing message with integration with path planning frameworks
- Known to be actively used
- Helper methods can be written to ease use

Cons:

- Relatively complicated

Goal Pose with geometry_msgs/PoseStamped

Pros:

- A very common message, very simple.
- Can be trivially upconverted to MultiDOFTrajectory with derivatives zeroed.

Cons:

- Maybe too simple

Odometry with nav_msgs/Odometry extended

Extend nav_mgs/Odometry to add acceleration

Publish the simpler version in parallel for backwards compatibility.

Pros:

- Supports needed acceleration estimates
- Based on successful message

Cons:

- Requires a new message

Adopt mav_msgs for velocity and accel

Pros:

- Well established messages been through several evolutions
- There are several existing implementations

Cons:

- Does not match ground robots interfaces

Other interfaces that could be reused

- Battery State via: sensor_msgs/BatteryState

Takeaways

- Standardization is important to allow parallel development
- Go through the process here as outlined yourself
- Make your own suggestions

I’m like to continue the conversation on ros-sig-mav@googlegroups.com

And make a proposal in a ROS Enhancement Proposal http://www.ros.org/reps/rep-0000.html

Please join the conversation!

Thanks